

Special Report
CMU/SEI-94-SR-06



Carnegie-Mellon University
Software Engineering Institute

AD-A283 367



Second Dependable
Software Technology Exchange

Charles B. Weinstock
Walter Helmerdinger

June 1994

DTIC
ELECTE
AUG 19 1994
S G D

DTIC QUALITY INSPECTED 1

94-26366



94 8 18 167

Carnegie Mellon University does not discriminate and Carnegie Mellon University is required not to discriminate in admission, employment or administration of its programs on the basis of race, color, national origin, sex or handicap in violation of Title VI of the Civil Rights Act of 1964, Title IX of the Educational Amendments of 1972 and Section 504 of the Rehabilitation Act of 1973 or other federal, state or local laws, or executive orders.

In addition, Carnegie Mellon University does not discriminate in admission, employment or administration of its programs on the basis of religion, creed, ancestry, belief, age, veteran status, sexual orientation or in violation of federal, state or local laws, or executive orders. While the federal government does continue to exclude gays, lesbians and bisexuals from receiving ROTC scholarships or serving in the military, ROTC classes on this campus are available to all students.

Inquiries concerning application of these statements should be directed to the Provost, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, Pa. 15213, telephone (412) 268-6684 or the Vice President for Enrollment, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, Pa. 15213, telephone (412) 268-2056.

Special Report

CMU/SEI-94-SR-06

June 1994

Second Dependable Software Technology Exchange



Charles B. Weinstock

Open Attribute Engineering Project

Walter Heimerdinger

Honeywell Technology Center

Accession For	
NTIS	CRA&I <input checked="" type="checkbox"/>
DTIC	TAB <input type="checkbox"/>
Unannounced <input type="checkbox"/>	
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and / or Special
A-1	

Approved for public release
Distribution unlimited.

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This report was prepared for the

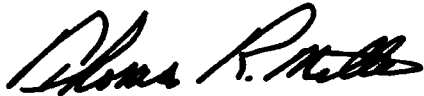
SEI Joint Program Office
HQ ESC/ENS
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

Review and Approval

This report has been reviewed and is approved for publication.

FOR THE COMMANDER



Thomas R. Miller, Lt Col, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

Copyright © 1994 by Carnegie Mellon University

This material may be reproduced by or for the U.S. Government pursuant to the copyright license under the clause at 52.227-7013.

This document is available through Research Access, Inc., 800 Vinial Street, Pittsburgh, PA 15212.
Phone: 1-800-685-6510. FAX: (412) 321-2994.

Copies of this document are available through the National Technical Information Service (NTIS). For information on ordering, please contact NTIS directly: National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161. Phone: (703) 487-4600.

This document is also available through the Defense Technical Information Center (DTIC). DTIC provides access to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential contractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center, Attn: FDRA, Cameron Station, Alexandria, VA 22304-6145. Phone: (703) 274-7633.

Use of any trademarks in this report is not intended in any way to infringe on the rights of the trademark holder.

Table of Contents

1	Introduction	1
2	In-Process Measurement	5
2.1	Technology Lecture: Dr. Ram Chillarege	5
2.2	Application Lecture: Mr. Michael Daskalatonokis	7
2.3	Panel Discussion	9
3	Software for Safety-Critical Systems	13
3.1	Application Lecture: Mr. Kevin Driscoll	13
3.2	Technology Lecture: Dr. John Knight	15
3.3	Panel Discussion	18
4	Software Testing and Reliability	21
4.1	Technology Lecture: Dr. Ravi Iyer	21
4.2	Application Lecture: Dr. Yitzak Levendel	23
4.3	Panel Discussion	25
5	The Tools Fair	29
6	Participants' Comments	31
7	Discussion	33
8	List of Attendees	35

Second Dependable Software Technology Exchange

Abstract: On March 24 and 25, 1994, the Open Attribute Engineering Project hosted the Second Dependable Software Technology Exchange. The exchange, sponsored by the Air Force Phillips Laboratories, the Office of Naval Research, and the Air Force Space and Missile Systems Center, brought together researchers and system developers, providing an opportunity for the researchers to learn the needs of the developers and for the developers to learn about techniques being investigated by the researchers. This report summarizes what transpired at the meeting.

1 Introduction

Dependability is important in avionics, vehicle control, logistics systems, and command, control and communications systems, to name but a few. As these systems increasingly rely on computers, software becomes more important to the dependability and safety of their users. Computer hardware has become more reliable, so software is now the bottleneck for achieving dependability. In recent years, however, a number of new approaches have emerged for building software for dependable system applications.

One way to facilitate technology transfer between researchers and practitioners is to get them talking to each other. With this mind, on March 18 and 19, 1994, the Software Engineering Institute's Dependable Real-Time Software Project hosted the first in what is hoped to be a series of Dependable Software Technology Exchanges. The second technology exchange was sponsored by the Air Force Space and Missile Systems Center and the Office of Naval Research.

The purpose of the exchange was to bring researchers and system builders together to provide an opportunity for technology transfer in both directions. System builders are often too busy to attend conferences or to write papers; researchers need exposure to "real-world" concerns. The technology exchange provided a forum in which each side could educate the other.

The exchange was coordinated by a steering committee consisting of

- Dr. Charles B. Weinstock, Software Engineering Institute
- Dr. Ravi Iyer, University of Illinois
- Dr. Walter L. Heimerdinger, Honeywell Technology Center
- Dr. Ram Chillarege, IBM T.J. Watson Research Center
- Dr. Fred Schneider, Cornell University

The focus was on three topic areas deemed critical for achieving dependability: in-process measurement, software for safety-critical systems, and software testing and reliability. Each area was allocated a half-day session divided into 3 parts. The session started with an hour-long presentation from a researcher followed by an-hour long presentation from an experienced practitioner. After a short break, a 75-minute panel discussion was held. Each panel consisted of the 2 speakers, plus up to 4 additional panelists. Audience participation was encouraged throughout the sessions. The technology exchange was videotaped.

A "tools fair" was held on the afternoon of Friday, March 25, with demonstrations by the Aerospace Corporation, AT&T Bell Laboratories, and the Software Engineering institute.

By all accounts, the Second Dependable Software Technology Exchange was well attended and successful. Of perhaps greater importance is the diversity of backgrounds represented by the attendees. The almost 60 attendees represented organizations including

Industry

- Allied-Signal
- AT&T
- Bellcore
- Boeing
- Cosgrove Computer Systems
- Eli Lilly and Company
- Honeywell
- International Business Machines
- Lockheed Missile and Space
- Loral Federal Systems
- Microsoft
- Motorola
- Raytheon
- System Technology Development Corp.
- Trident Systems
- Union Switch & Signal
- UniView Systems

Academia

- Carnegie Mellon University
- Cornell University
- Ellendel/Stockholm University
- University of California
- University of Cincinnati

- University of Illinois
- University of Virginia
- University of Texas
- University of York
- Wheeling Jesuit College

Government

- Air Force Office of Scientific Research (AFOSR)
- Federal Aviation Administration (FAA)
- Institute for Defense Analyses (IDA)
- National Institute of Standards and Technology (NIST)
- Naval Surface Warfare Center (NSWC)
- Office of Naval Research (ONR)
- Air Force Safety Center
- Phillips Laboratories
- Sandia National Laboratories

Other

- Aerospace Corporation
- MITRE
- Software Engineering Institute
- SRI International

We clearly achieved our goal of bringing researchers and practitioners together.

At the end of the technology exchange, attendees completed a feedback form to help us evaluate the success of the meeting. Although not every respondent found all of the talks useful, every talk was useful to some of the respondents. Virtually every respondent was enthusiastic about holding further technology exchanges—a variety of topics were suggested.

The next sections describe the three sessions in detail, including the speakers and panelists as well as a summary of each discussion. Following those sections is a synopsis of the comments from participants. A complete list of attendees appears as an appendix.

2 In-Process Measurement

The morning session of Thursday, March 24, 1993 focused on in-process measurement. Dr. Ram Chillarege from IBM's T.J. Watson Research Center gave the technology lecture and Mr. Michael Daskalatonakis of Motorola's Cellular Infrastructure Group gave the application lecture. In addition to these speakers, the panel consisted of Dr. Michael Lyu of Bellcore, Mr. Tom Gilchrist of Boeing Commercial Aircraft, and Mr. Roger Sherman, director of testing at Microsoft. Dr. Chillarege coordinated the session.

2.1 Technology Lecture: Dr. Ram Chillarege

Chillarege began his lecture by stating the goal of defect-related in-process measurement and analysis: to identify areas in a product that need corrective action while the product is *still in development*. He noted that the step in which defect information is fed back into the software product development process is the weakest part of the software quality life cycle today. He observed that process measurement and analysis spans a spectrum from statistical defect models that are abstract, mathematical, and often not relevant to the needs of the programmer to root-cause analysis that is qualitative, down-to-earth, and represents the programmer's perspective. Chillarege then introduced orthogonal defect classification (ODC), an approach that brings the extremes together.

He noted that software defects have a life cycle that begins with the injection of an error. The error escapes inspection and then escapes testing, remaining dormant until it creates a fault, which leads to a failure. The defect defies identification until a "fix" is incorporated into the product. He noted that defect data is collected at each stage in this life cycle, but the data is neither uniform nor consistent because the classifications used vary depending on the stage in which the data is collected. He noted that defects discovered in software inspections are usually classified as major or minor. Accurate defect data is rarely available from unit test activities, as programmers are reluctant to provide data from this activity — often defect statisticians guess data in this stage. Functional testing usually yields good data, as does system testing. Defects discovered in these stages are usually classified in terms of three or four severity levels. Field data uses a different classification, usually based on severity.

Chillarege summarized the traditional approach for defect analysis that deals only with the number of defects, analyzing either cumulative defects over time or defect density over time. In this approach, the analyst searches for a knee in the defect curve that supposedly indicates the point at which few new defects are found and the product is "mature." However, the knee of the curve may only indicate the cessation of testing, and a significant number of defects may remain in the product.

He contrasted this approach with an alternative analysis approach that measures progress by examining the distribution of defects, thereby extracting a "process signature." This approach is embodied in ODC, which he defined as "a semantic classification on defects, such that they collectively (in their distribution) explain the progress of the product through the process." Furthermore, the distribution can also point to the part of the process needing attention.

Chillarege explained that ODC requires a set of defect classes that can be related to the process, and therefore used to explain the progress of the product through the process. Additionally, the set of all values of defect attributes should form a spanning set over the process subspace. He then explained how ODC achieves this by grouping defect attributes into five attribute classes: defect type, trigger, source, environment, and impact.

The *defect type* attribute is associated with activities in the software process. Typical types are missing and/or incorrect: function, interface, checking, assignment, timing/serialization, build/packing/merging, documentation, and algorithms.

The *trigger* attribute relates to the condition that forces a defect to surface (the catalyst for detecting a defect). Typical triggers include: workload/stress, recovery/exception conditions, startup/reset, timing, hardware configuration, software configuration, or execution mode. Other triggers are boundary conditions, bug fixes, and customer code.

Source type attributes might include: old functions, new functions, reused code, rewritten code, repaired code, vendor-written code, or scaffolded code. Other source types may identify particular customer sites.

Impact type attributes gauge the effect of the defect. They might include: capability, usability, performance, reliability, installability, maintainability, documentation, integrity/security, and standards. The first seven criteria, referred to by their initial letters CUPRIMD, are used in IBM.

Chillarege then discussed how triggers can indicate the effectiveness of the various inspection and test activities. For example, if the distribution of triggers from a system test does not mimic the distribution from field reports, the system test process should be revisited. He noted how trigger distributions from reviews or inspections can be used to assess the skill of the inspectors. He recounted a situation in which data from an inspection of a major product showed very few interface defects, indicating a lack of interface skills in the inspection team. A subsequent inspection of the product by an experienced analyst uncovered over 200 interface defects.

Chillarege showed how associations between defect type and process stage can be used to gauge product maturity. He illustrated how function defect data can be fed back into the design process by exposing weaknesses in the various life-cycle stages and by indicating the need to revamp a particular stage of a product.

Chillarege concluded by noting that a defect control program could complement a defect prevention program. Hot spots discovered in the defect control program could be used to launch the defect prevention program in the right direction. Once the prevention program has taken an action, the defection control program can measure the response and use the experience to locate similar defects.

2.2 Application Lecture: Mr. Michael Daskalatonokis

Michael Daskalatonokis of the Motorola, Inc. Cellular Infrastructure Group delivered the application lecture on in-process measurement. He began by emphasizing that measurement is not a goal in itself — the real goal is to make the software development process faster (reduced cycle time) and cheaper (greater productivity) and to make the product better (higher quality).

Daskalatonokis noted that metrics at Motorola are targeted at individual audiences that each have specific interests. For example, senior managers are interested in overall improvement across projects, software managers are more interested in improvements in their individual projects, and individual software engineers are interested in improving specific project activities and work projects. Historically the Motorola metrics program began at the corporate level and migrated down to the project level.

Daskalatonokis showed metrics charts aimed at executives that summarized five sets of data: (1) *software process and product quality*, as measured by in-process faults and defects per thousand lines of assembly-equivalent code; (2) *customer satisfaction*, as measured by *customer-defined* attributes, such as features, reliability, availability, billing, and documentation; (3) *cycle time/productivity*, measured by the calendar time required to develop new or enhanced software and code produced per person-month; (4) *progress in implementing software engineering technology*, measured by a multiaccess comparison of progress toward goals in project management technology, development technology, etc.; and (5) *progress in meeting the Motorola corporate goals of SEI software capability maturity level 3 by 1995 and level 5 by 1998*, measured by progress in each of the SEI key process areas.

Daskalatonokis then showed 10 sample charts that would be used by the quality managers of a Motorola division. Each of the 10 sample graphs showed one or more software metrics for a 5-year period. The first 3 charts dealt with software defects — in-process software defects, total released defects, and defects found by customers, displayed on a logarithmic scale that made it easy to track quality in terms of sigma. Daskalatonokis noted that Motorola had adopted a corporate goal of attaining a 6-sigma quality level (3.4 defects/million units) in both hardware and software. Other sample charts displayed the number of post-release problem reports (both new and cumulative), post-release problem report aging (the average cycle time required to close out both open and closed problem reports), cost to fix post-release problems, total defect containment effectiveness (the portion of the total faults in a software product life cycle found before customer release), the defect containment effectiveness of the major de-

velopment phases (requirements definition, high-level design, low-level design, and coding), the ratio between estimates of schedule and effort and actual values, and both incremental and cumulative productivity in K-lines of code per person-month.

Next, Daskalatonokis showed examples of metrics that could be used to plan and control each of the major software life-cycle processes: project management, requirements definition, design, coding, inspections/reviews, and testing. Daskalatonokis exhibited a number of in-process metrics, while emphasizing that a given project with Motorola would use only a small subset of the metrics presented (and possibly some metrics not presented).

The sample management metrics displayed a timeline of items completed (development phases or modules) or resources used (personnel, earned value, disk space, or target systems). Much of this data is collected automatically (staffing information is extracted from engineer time records, disk usage from the file system, target system usage through e-mail from an automated collecting tool).

Requirements metrics included timelines of the number of requirements and the number of changes to requirements during a project. Such metrics are in use for Iridium, a project large enough to provide meaningful results. Other sample requirements metrics track the type of problem leading to changes (omission, commission) and the source of changes (marketing, customers, system developers, or software developers).

Metrics for coding are not widely used by Motorola. Daskalatonokis discussed the relative system complexity (RSC) metric and McCabe's cyclomatic complexity metric, which are not used much at Motorola, and Halstead's software science, which is used in a few areas in Motorola (i.e. the Paging Division) with good accuracy in estimating defects to be found.

Daskalatonokis's sample metrics for software inspections focused on the results (density of faults found) and the productivity (number of lines of source code per hour) of individual inspections. Both of these charts must be interpreted using limits based on experience. Other sample inspection metrics dealt with rework cost savings, fault severity, and the reasons for faults found in code inspections.

Testing metrics considered both testing effort, measured in the number of test cases estimated and completed, and testing results, measured in terms of the number of failing test cases. Daskalatonokis discussed the Ralph Brettschneider criterion for releasing software, a model used widely by Motorola, especially in the communications sector. The model, discussed in more detail in a July, 1992 issue of *IEEE Software*¹, estimates the number of hours that testers should continue testing without finding defects to achieve a desired number of post-release defects, given the number of defects found to date and the total test execution hours to date.

¹. Sheldon, F.T., et. al. "Reliability Measurement: From Theory to Practice," *IEEE Software* (July 1992): 13-20.

In summarizing his experiences and lessons learned at Motorola, Daskalatonokis discussed a number of obstacles to be overcome in an in-process metrics program, including an initial lack of tools to automate metrics, fear of extra cost and overhead, fear that data will be used as a weapon, especially if other projects do not report data consistently, and a lack of expertise on the use of metrics for in-process planning and control. Factors critical to the success of an in-process metrics program include: gaining senior management support, addressing the needs of all metrics audiences, allowing groups to tailor their own metrics, using a staged implementation approach with multiple short-term projects as pilots, identifying a champion for the metrics effort, providing ongoing training and consulting, and extracting metrics from tools that are already used by projects. The essential element is that engineers and managers must be the owners and users of the metrics data. This can be done by providing them with sample reports, encouraging them to create their own reports, and then revising and automating these reports. If this is done, in-process metrics can help improve project planning and tracking and can establish a historical baseline for long-term improvement.

Daskalatonokis concluded by re-emphasizing that measurement is not the goal—the goal is improvement through measurement, analysis, and feedback.

2.3 Panel Discussion

The panel discussion period began with short presentations from each of the panelists that were not lecturers.

Michael Lyu presented nine obstacles to software quality:

1. Quality is perceived as a cost, not as value.
2. Software quality is accorded a low priority because features and functions bring revenue.
3. Customers have a high tolerance of poor software quality.
4. There is a lack of a "problem"—a crisis is needed before improvement initiatives can be taken.
5. There is a lack of reasonable and verifiable software quality specifications derived from user requirements.
6. The pursuit of quality delays product schedule and increases cost.
7. There is high turnover in the permanent staff that works on improvement activities.
8. Data collection consumes resources and is confusing, offensive, and error-prone.
9. Intrusive monitoring is unacceptable, but monitoring without intrusion is impossible.

He then posed nine issues and questions:

1. Can we learn from other industries (e.g. automotive)?
2. How to tailor the measurement processes implemented in other companies.
3. How to define software quality metrics that make sense to managers.
4. How to collect data accurately and painlessly.
5. How to analyze and interpret data to gain useful insights.
6. How to make continuous quality improvement a cost-effective process.
7. Should we settle down for continuous improvement or search for major leaps?
8. What techniques have proved to significantly improve software quality?
9. How to attract the best people to do measurement work.

Tom Gilchrist of Boeing began with an observation that a major problem is the disagreement between management and the people doing the work and creating the value.

He reiterated two quotes on systems and organization. The first, by Tom Gilb: "A system tends to grow in terms of complexity rather than simplification until the resulting unreliability becomes intolerable." The second by Ron Nelson: "Left to their own, organizations tend to migrate toward individual practice."

Gilchrist then contrasted measurement without feedback (which he characterized as "management for the curious") with control without measurement (which he characterized as "royal dictates").

He then presented two models of the software development process: a single-engine model and a two-engine model. The single-engine model follows a process that converts needs and constraints into deliverables. Action in this process is induced by a cycle that compares exit criteria with values under the influence of triggers, business objectives, and constraints. This model has no memory and can be uncontrollable if cycle times are longer than 4-6 weeks. The two-engine model has an engine to maintain and improve the process coupled with an engine to use the process.

Finally Gilchrist observed that people who *collect* the metrics must *use* the metrics, otherwise a situation may occur in which people write programs to add lines of code to their products.

Roger Sherman of Microsoft stressed three points. First, he mentioned the importance of reliability. He stated that although no one ever died from a bug in Microsoft's Flight Simulator, Microsoft is driven by profitability, and a recall of a product to fix a bug can destroy profit, especially if the product is widely distributed, like Windows, with 55 million copies in use. Next, he stressed the importance of process. He contrasted the measurement of process with the measurement of the state of product. He noted new trends in software development, including larger teams (200 people for Excel/Word, 75 of which are developers) and developers who are unlike the users they serve (60% of Microsoft sales are outside of the US).

Finally, Sherman outlined the use of metrics at Microsoft. He noted that product teams are almost completely autonomous, even to the point that they have their own process, and that senior management generally does not access metrics.

The panel then addressed questions from the audience.

Roy Maxion of Carnegie Mellon University observed that most obstacles are social and asked how they should be addressed. Sherman replied that it is important to define your goals, as quality means different things to different people. Lyu added that cultures change with time and that culture shouldn't be allowed to hide technical excellence. Gilchrist claimed that the major issue is goals — how many "Sunday school truths" (truths observed only on Sunday) are in an organization. He noted that goals are complex. He also recalled that Deming states that a customer relationship must be win-win — artifacts don't make the difference if attitudes do not change. 100% inspection (and test) is not enough.

Michelle Hugue of Trident Systems commented on work with a major accounting firm known for its expertise in quality that encountered obstacles that could not be surmounted, in part because of too many hidden agendas.

Yitzak Levendel of AT&T countered that the problem is technological, not social. Because we don't have the technology, we don't know how to separate human behavior from the machinery. Michael Daskalatonokis replied that cultural issues still make a difference in adopting metrics.

Ravi Iyer commented that he sees major issues in high-speed communications software, but doesn't hear of any technical issues. Chillarege replied that every effort to reduce software development to a "mechanical" activity (by reducing variance etc.) has not been successful because software is not the same as automobile production. He stated that he knew of two cases that reached the 6-sigma quality level (3.4 errors per million source lines of code). Chillarege stated that we don't know the "thermodynamics" of software because we do not have a model yet. Nevertheless, metrics still provide useful insight. He stated that practitioners are seeking a way to produce value in a business in which the tools to produce software in a short time are improving rapidly.

3 Software for Safety-Critical Systems

The afternoon session of Thursday, March 24, 1994 focused on the problem of developing software for safety-critical systems. Mr. Kevin Driscoll of the Honeywell Technology Center gave the application lecture. The title of his talk was "A View from the Trenches—Software War Stories." Prof. John Knight from the University of Virginia gave the technology lecture. The title of his talk was "Safety-Critical Software Technology Issues." In addition to these speakers, the panel consisted of Dr. Walter L. Heimerdinger of the Honeywell Technology Center, Mr. Mike DeWalt of the Federal Aviation Administration, Mr. Lynn Elliott of Cardiac Pacemakers, Inc., and Prof. John McDermid of the University of York. The session was coordinated by Dr. Heimerdinger.

3.1 Application Lecture: Mr. Kevin Driscoll

Developing safe software is a difficult business. This is especially true for the control systems that must interact with an environment by sampling inputs and periodically emitting outputs in accordance with some time base.

According to Driscoll the five common causes of unsafe software are

1. Incomplete requirements
2. Non-deterministic design
3. Inadequate testing
4. Improper use
5. Interdisciplinary "cracks"

Driscoll's discussed these five points.

Driscoll noted that a design at a particular level of detail provides requirements for the next lower level of detail, with each successive level linked by rationales and compliance audits. Echoing the requirements talk at the previous Dependable Software Technology Exchange, Driscoll said that the major difficulty with requirements is that all of the assumptions aren't written down, particularly assumptions about the physical world. This can lead to situations in which the designer makes one set of assumptions and the programmer or ultimate user makes another conflicting set of assumptions. Furthermore, given the ambiguity inherent in English, even explicit requirements are open to misinterpretation. In either case the result can be an unsafe system.

A deterministic design makes it possible to understand what a system will do (that is, what perceptible outputs will be produced) in the presence of certain inputs. In a non-deterministic system, a system's future behavior cannot be accurately predicted from its current behavior. Driscoll claims that an unpredictable system cannot be called a safe system.

Driscoll feels that most testing is inadequate because it doesn't look for unintended functionality. Unintended functionality is usually latent. Of course, testing for all possible unintended functionality would require an infinite amount of testing. As an example, Driscoll illustrated the misuse of a SETBIT function in Fortran. The designer and implementer had different views on how the function was to be used, and the result was that an unintended bit in memory was being set and cleared. Initially this went undetected because the bit was in an "uninteresting" location in memory. When some totally unrelated code that changed the distance to the target of the SETBIT function was modified, the bit ended up in an "interesting" area, and the system failed.

Driscoll said that testing is good at catching permanent faults, but less effective at dealing with intermittent and latent faults. Low probability faults are difficult to find but can occur often enough in a large installed base to be a significant problem. As an example Driscoll cited the case of an autopilot that would intermittently jump in altitude every couple of years. Over the installed base, there were three incidents per month. They never found the problem.

Driscoll also used an autopilot example to illustrate how misuse can make a system unsafe. Pilots tend to rely too much on the autopilot, using it to climb or descend instead of flying the plane themselves. This was not the intended use for the autopilot. When the operators place more reliance on a digital system than was intended, the result can be an unsafe system. This raises the question of whether the designer should be responsible for the new uses of the application.

As another example, related to misuse, Driscoll cited the human interface on the Airbus A320 in which a crash was traced to the pilots' confusion between vertical speed and the flight path angle.

Driscoll noted that the apparent dichotomy between synchronous and asynchronous systems is not as sharp as might appear; instead synchronism can appear at a number of levels, ranging from the micro-cycle clock level to the instruction level and higher levels, such as major frames and external events. According to Driscoll asynchronous designs should never be used for safe systems. In particular he feels that special care must be taken when software interacts with hardware; for this reason interrupts should be avoided. As an example of the problems this can lead to, Driscoll showed how updating a linked list the "wrong way" in the presence of interrupts could lead to an infinite loop. In the system from which this example was taken, the infinite loop was in the microcode, and it was necessary to remove power from the system to perform a reset. Because of the asynchronous nature of the system, this occurred only about once a day and was extremely difficult to find. Driscoll had several other examples of how interrupts and asynchronism could lead to unsafe systems.

Driscoll asserted that the system designer also must know the application environment. As an example he showed the calculations needed to compare the outputs of two sensors for α , the aerodynamic angle of attack of an aircraft. One of the principle reasons for using two sensors is to be able to detect a failure in one of the sensors by noting when the difference between the values provided by the sensors exceeds an error threshold. Unfortunately, the value for an

appropriate error threshold is not constant; instead it will depend on a number of conditions, including the location of the aerodynamic center of the aircraft (which itself is not a constant), the pitch rate, the radius of each sensor from the aerodynamic sensor, etc. A software engineer without extensive domain knowledge would not be able to make this determination accurately.

Driscoll concluded by emphasizing that system safety rather than software safety must be the goal. This can be achieved only if a system is built by "real" systems designer/architects—people with knowledge of a number disciplines. They are needed to compensate for narrow specialists who do not communicate adequately.

After reminding the audience that most catastrophic failures result from the hardware/software interactions and design/specification errors, he reviewed contemporary practices that impede the development of safe software:

- Incomplete and constantly changing specifications.
- The misconception that safety is a result of quality, which can be realized by sufficient testing.
- The lack of a central system architecture for most systems.
- Poor communications between the numerous disciplines required to implement a system.
- The inevitable cost and schedule constraints under which most systems are developed.

3.2 Technology Lecture: Dr. John Knight

Knight began his lecture by observing that safety-critical applications are everywhere. In addition to the commonly thought-of nuclear, medical, and aerospace applications, Knight claimed that we must broaden the definition of "safety" and add financial, service, and consumer applications. To emphasize this point, he discussed a failure at the Bank of New York that shut down bond trading and caused the bank to lose a lot of money. This was certainly unsafe to the health of the bank. Yet, he noted, we have a very difficult time building systems that meet the required level of assurance. For instance the often-cited requirement that the probability of failure must be less than 10^{-9} per hour of operation cannot routinely be met. Yet, in many situations this is a reasonable requirement, because these systems are replacing hardware systems that were that good, and we're not prepared to have a system that is less safe.

Knight echoed Driscoll and claimed that specification is the first problem. Most errors are *caused* at the front end of the life cycle. However, most errors are *found* at the rear end of the life cycle. This suggests that software engineers are good at what they do and that many problems are created in the specification process. Especially difficult is when the customer revises the requirements toward the end of the development.

Knight described two important roles in the implementation of a safety-critical system—*applications engineers* understand the functions required of the system as well as the attendant hazards and risks, while *software engineers* understand the structuring, implementation, and analysis of software. He asserted that software engineers, while good at what they do, are not qualified to answer applications engineering questions. We certainly don't want the software engineer to decide what to do with an unspecified input. Hazards, risks, etc., must be handled very carefully in software specification. Knight says that if we are not extremely careful, something will fall through the cracks and the result can be catastrophic.

What we need, according to Knight, is to use more precise specifications. This requires something more than natural language, and, as John Rushby discussed at the previous exchange, formal (or semi-formal) languages are an appropriate choice. Data flow diagrams, Z, and Vienna Development Method (VDM) are all examples of ways to communicate more formally than with English.

But, Knight said, there is a problem. Although engineers accept formal notations like "C" and the calculus, they are not quick to adapt to formal mathematics. At least partially, this is caused by a lack of production-quality tools to support formal methods. Furthermore, the fact that there is no single notation that is comprehensive requires the user to understand and use several different notations. Also, certain things are extremely difficult to specify including realtime, computer arithmetic, and user interfaces. But, Knight feels that formal specifications are the best hope, the techniques are not that difficult to apply, and there already is an extensive base of industrial experience in the use of formal methods including the Darlington Nuclear Generating System, the Paris Metro signaling system, and the Traffic Collision and Avoidance System (TCAS).

Knight then recounted an approach followed at the University of Virginia in which a formal notation (Z) was selected and used to document the results of a system fault tree analysis. Unfortunately, the result was unsatisfactory—an ad hoc process with no guarantee of repeatability. The team then developed an approach to separating software concerns and systems engineering concerns and to separating software safety and specification safety by modeling the flow of information through the overall system. As a result, Knight felt that they have developed a process that is rigorous, repeatable, and dependable, that works with real-world problems, and that applies to a wide spectrum of systems at many levels of formality.

Knight then went on to talk about the design and implementation phases of developing safe software. He started by discussing how reuse can provide more benefits in terms of increased dependability than it provides toward the more traditional goal of increased productivity. The key idea is that if we reuse something, we don't have to verify it again *if we are careful*. This means that spending a lot of time verifying it the first time might be cost effective. This only works with systematically planned reuse, not with opportunistic reuse, and is likely limited to reuse in a particular domain. He noted that systematic reuse must span the entire software life cycle, including specifications.

According to Knight, the programming language chosen can have a big impact on system safety. Even if we verify the source code, we are still dependent on the compiler, libraries, linker, and runtime system. Knight contrasted Ada 83 and C++. Ada has many language benefits for safety-critical applications including type checking, use restrictions, constraint checking, etc. But Ada has holes such as exception handling and precise and rigorous, but not very useful, time semantics. Ada 9X will solve some of the existing problems, but the result still won't be perfect for safety-critical system development.

C++ is another popular language that, according to Knight, should not be used in any application requiring dependability. There are several problems. Although a very powerful language, C++ is based on C. C has some very insecure properties such as its significant dependence on pointers, and relatively poor type checking. Pointers encourage programmers to "throw around" machine addresses as variables. Furthermore, Knight said, a great deal is hidden from the programmer, including the use of constructors and virtual function resolution. There is no precise definition, and there is never likely to be one. On the other hand, the program-structuring benefits of the class mechanism are considerable.

Knight went on to discuss verification through testing and some of the problems that occur. Many systems are difficult to test at best. For instance, graphical user interfaces present real problems if humans have to do the testing manually. Repeatability becomes a problem. Real-time systems are difficult to test without disturbing the system, and the tests are difficult to reproduce. Distributed systems suffer from many of the same difficulties. In general, Knight says, we have a problem telling whether an output is correct and whether the test input profile matches the operational profile. Knight also raised the issue of nondeterminism.

According to Knight verification through inspection is a very powerful tool. Inspection techniques include formal inspections, active design reviews, and phased inspections. The data is so overwhelming that inspection is both effective and cost effective and works in all of the life-cycle phases.

Another technique that Knight discussed was verification through proof. There are simple but useful properties that can be proven very easily, such as the absence of infinite loops, freedom from some forms of deadlock, etc. The key here is that if you can prove it, you don't need to test for it.

Knight had some criticisms of the role that standards play in dependable systems. The first criticism is that there are too many standards. According to Knight, we should ask ourselves

- Who wrote the standards and were they divinely inspired?
- What does the standard mean (i.e., is it precise enough?)
- Has the standard been tested?
- How can one tell if one has complied with the standard?
- What does compliance really buy?

Knight gave some examples of problems with standards based on the United Kingdom's MOD-00-55 standard.

Knight closed his presentation with a discussion of dependability assessment. Here he talked about two concepts, life testing and reliability growth models. Most life testing is based on models for hardware that don't exist for software. For ultra-dependable systems, such assessment is infeasible (it takes too long to observe a system that has a failure probability of 10^{-9}). Worse still, Knight said, current software models presume perfect error detection. Not noticing that a system has failed during a test makes the problem much worse and leads to reliability estimates that are much too high.

Assessment through reliability growth models involves executing a program until it fails, repairing it, and the repeating the process, keeping track of mean time between failures (MTBF), which presumably increases. Knight says that this is an interesting technology but that its utility is unclear.

3.3 Panel Discussion

Following the presentations there was a free-form panel discussion with much interaction with the audience. Presentations by the panelists were eliminated to provide additional time for audience interaction.

Ram Chillarege asked the panelists what they thought were the current best practices in software safety. The general consensus was that safety analysis should be performed at the system level, not at the component level, and that domain experts were necessary. Also, much emphasis is placed on simulations. Mike Dewalt mentioned that the FAA is trying to change emphasis from process (e.g., DO178B) to the analysis of critical system properties.

In a brief interchange regarding reuse, it was pointed out that there is no significant reuse at the architecture level. It was also pointed out that reuse often presents problems at interfaces unless the system was designed with reuse in mind.

John Knight followed up his comments about C++ with some specific examples of why he considers C++ to be a poor choice for safety-critical software. Although it does many things well, there are some constructs that are extremely error prone [for example, writing "if (a = b)" when what was really meant was "if (a == b)"]. Ada 9X doesn't solve the problems. It has only single inheritance, and it hasn't fixed the Ada exception-handling problems. Art Robinson proposed using an Ada subset, but the consensus seemed to be that this was too limited for most applications. It was pointed out that there have been no airworthiness directives issued that could be traced to compiler or language problems. It was also pointed out that if you design properly up front, it probably doesn't matter what language is used.

John Rushby then turned to formal methods and pointed out that an early commitment to formal methods means that you have a description early in the life cycle that can be analyzed. John Knight talked about the difference between formal methods and formal notations. People

often refer to a notation as a method. Formal specifications make us think carefully, which results in a means of communication. Mike Dewalt discussed Nancy Leveson's TCAS example and mentioned that she was unable to derive safety properties from the formal specification. Rushby was skeptical of the social benefit of formal notation without analysis. Michelle Hugue mentioned that formal methods can be used as a discovery tool.

Stephen Johnson turned the discussion toward system-level concerns. In particular, he was concerned that most people do hardware or do software, but do not take a system-level approach to dependability.

Chuck Weinstock asked Lynn Elliott how safety practices differ between the aerospace community and the medical electronics community. He responded that availability was a key feature of the systems with which he deals. The system may sit dormant for months at a time and then be required to give the heart a jolt on a moment's notice. They spend a lot of time simulating various situations. Since their products are embedded in the human body, they have to contend with the fact that each body presents a different environment. There are also severe physical space limitations. The software for the system Elliott described is about 75,000 lines of code.

Elliott also pointed out that they need a low defect rate and can't reasonably recall the product. This has led them to reduce complexity by removing features that are not absolutely required. Surprisingly there is no formal fault tolerance in the system (beyond built-in memory checking), in part because of the need to minimize the amount of code and consequent energy (battery) consumption.

Joe Marshall then turned the discussion to the asynchronous versus synchronous issue and asked about the place of interrupts in a safety-critical system. Kevin Driscoll reiterated that he believed that there was no place for interrupts in a safety-critical system. He feels that there should be (at most) two interrupts: the clock and a fatal interrupt. John Knight pointed out that the main problem with interrupts is that they result in a context switch that makes analysis difficult. No interrupts leads to simpler analysis, but there are tradeoffs. Synchronous systems are harder to design.

For the last question, the panel was asked when they thought that the 10^{-9} failure probability would be achieved. Kevin Driscoll said that it was a matter of demonstration only. Walt Heimerdinger and Mike Dewalt said that the number is meaningless for software (but reasonable for hardware). Dewalt says that we are lucky to support 10^{-6} to 10^{-7} . Knight reiterated that the 10^{-9} originated from the fact that we were replacing hardware with software. He feels that there is plenty of room for expert systems, etc., which don't have to meet the 10^{-9} failure rate.

4 Software Testing and Reliability

The morning session of Friday, March 25, 1994 focused on software testing and reliability. Dr. Ravi Iyer of the University of Illinois gave the technology lecture. The title of his talk was "Issues in Software Testing and Reliability." Dr. Yitzak Levendel of AT&T Bell Laboratories gave the application lecture, "A Success Story: Ten Years of Software Quality Improvements or the Tunnel at the End of the Tunnel." In addition to the speakers, the panel consisted of Dr. Michael Dyer of Loral Federal Systems, Mr. Bliss Jensen of AT&T, Dr. Steve Bunch of Motorola, and Dr. Jacob Abraham of the University of Texas at Austin. This topic area was coordinated by Dr. Iyer.

4.1 Technology Lecture: Dr. Ravi Iyer

Iyer began his talk by contrasting hardware testing with software testing. He noted that hardware testing is primarily concerned with physical faults. Hardware fault models, he said, are simple, proven, and are supported by public domain tools that have been validated by years of experience, while software faults are design faults for which there are no generally accepted fault models. Iyer observed that software development typically begins with a base of millions of lines of code and adds hundreds of thousands of lines of additional code for each new feature implemented. He quoted an estimate by an official of Boeing that the Boeing 777 airplane would fly with over 4 million lines of code. He stated that the challenge in such an environment is to implement changes while maintaining software quality and remaining competitive in both cost and time to market. The usual approach is to focus on testing, relying heavily on regression testing to maintain existing quality and attempting to maximize test efficiency in terms of coverage, cost, and automation.

Iyer reminded the audience of the increased costs associated with finding an error later in the software life cycle. Not only does testing become more complex and costly at later stages in the development cycle, but change documentation becomes more widespread and costly, more people are involved in locating the problem and making changes, and more regression testing is required. Furthermore, once system operation begins, the development team may no longer be available. Iyer illustrated how the quality of a software product depends on the user in the case of a telephone switching system. Subscribers expect available dial tones, reliable connections, and no lost calls; operating staff expects to install new features without service disruption while minimizing human error; accountants are concerned with billing accuracy, and maintenance programmers need good documentation, software that conforms to specification, and dependable repair methods and tools.

Iyer began a discussion of approaches for achieving quality software by listing a number of techniques for defect prevention, including top-down design and implementation, bottom-up design and implementation, modular design and programming, structure design and programming, defensive design and programming, automatic programming, design representations, design and programming tools, and formal methods/proofs. He singled out cleanroom soft-

ware engineering as a radically different approach that is claimed by some to be highly efficient. Bugs are removed by analyzing the software with correctness proofs instead of execution tests — no execution of the software takes place until the software is deemed ready for shipment. Thus, testing is used to certify that the software product meets requirements instead of looking for bugs.

Iyer then addressed the second approach for achieving quality software, defect removal. This approach includes both passive methods and active methods. Examples of passive methods include: requirements specification, design or code reviews, pseudolanguage analysis, formal verification, and symbolic execution. Testing is an example of an active method. He reviewed the hierarchy of tests for a telephone application, with white box testing (unit test and intra-feature test) at the bottom of the hierarchy, followed by black box testing (integration and inter-feature test), system and network level test, performance and capacity test, first office application test, and customer acceptance test.

Iyer contrasted the two primary software testing methods, functional testing and structural testing. He noted that functional testing checks for defects in computation (overflow, underflow, loss of accuracy), data handling, interfaces, concurrency, synchronization, and for performance at specified levels and under stress. Structural testing includes path testing, data-flow testing, integration testing, and mutation testing. Path tests are selected to exercise every statement or instruction at least once, exercise each option of every branch or case statement at least once, or exercise every path from entry to exit. Data-flow testing selects test paths in a program's control flow based on the definition and user of data objects.

Integration testing tests for errors not discovered during unit testing, including missing functions, extra functions, and interface errors. Iyer mentioned an experiment on integration conducted by Solheim in 1993 that analyzed the number of defective modules remaining in a set of 15 artificial software systems. He also discussed the results presented in a classic paper by V. Basili in 1987² that compared the number of faults detected in four different programs by 74 subjects ranging from advanced to junior, using three testing strategies (code reading, functional testing, and structural testing).

Iyer closed with a discussion of the effectiveness of testing as measured by subsequent failures in the field. Specifically, he discussed a study of the AT&T 5ESS system conducted by the University of Illinois, Urbana Campus. Data from eight testing teams was used to build a database of test, fault, repair, and software functionality. The effort stemmed from observations that the results of many test teams were not always correlated with results from the field. The objective was the reduction of testing cost by improving test selection and eliminating redundant tests. An analysis of this database showed that testing effectiveness (the ratio of repairs to test runs) was 2%, testing redundancy (the ratio of the number of failed tests to the minimum number of tests needed to find the faults) ranged from 4% to 16%, and fault record effectiveness (the ratio of product repairs to the number of fault reports written) was 35%, in-

² Basili, V.R. & Selby, R.W. "Comparing the Effectiveness of Software Testing Strategies." *IEEE Transactions on Software Engineering* (December 1987): 1278-96.

dicating that each fix involved about 3 faults. A white box analysis was used to identify the subsystems producing the most errors to guide future testing efforts. Efforts are now under way to automate this procedure.

4.2 Application Lecture: Dr. Yitzak Levendel

Levendel began his lecture with a caveat that he would address only continuous-operation, real-time systems. He structured his lecture around his experiences during a decade of software dependability improvements in telephone switching systems, principally the 5ESS switch. After observing that he would define a fault as "whatever we end up fixing" and that data tends to be perverted (i.e., multiple repairs are assigned to one problem), he showed how the outages per 10,000 lines had decreased substantially from the 5E4 system (5ESS, release 4) to the 5E8 system. He presented 1992 data that contrasted 5ESS switch downtime of 1.6 minutes per 10,000 lines in service with comparable values of 6.3, 20.2, and 28.7 for competing products.

Levendel then outlined nine problems behind the story of dependability improvement. First, he remarked that testers of an early release (5E3) noted that few new failures were encountered after the test program reached a certain point, leading to a premature convergence of the quality metric to acceptable levels. It was subsequently discovered that the test program consisted of a fixed set of tests that were rerun repeatedly; hence as time passed, most of the tests were reruns of successful tests. More accurate test data was obtained by filtering out the results of these reruns and concentrating on known problem areas in the system under test.

Second, Levendel displayed a cumulative failure rate curve with a noticeable upward change in the failure rate after thousands of tests had been completed. He attributed this phenomenon to excessive reliance on predefined test programs that are not capable of detecting failures due to new conditions in the system. Levendel stated that the solution in this case was to use statistical sampling techniques. If the number of passes in the first pass of 10-20 tests in a given area is above an upper confidence limit, the area is accepted. If the number is below a lower confidence limit, the area is rejected. Otherwise testing is continued until a subsequent pass produces results beyond one of the two limits. Levendel observed that the length of time between the test passes is important: two weeks worked best at AT&T.

Third, Levendel showed two cumulative test pass rate curves; one with a higher constant pass rate than the other. He observed that the quality of the underlying product was the same in both cases; the curves were really measuring the ability of an organization to fix defects (the higher curve was related to a faster fix turnaround time than the lower curve). As a result, Levendel has dropped cumulative pass rate as a measure of product defects remaining and has moved to a less "tamperable" metric, the failure *opening rate*.

Fourth, Levendel observed that statistical testing covers a very small percentage of the software in a system. To provide more useful testing results, he recommended keeping a database of tests vs. modules tested to pinpoint weak modules. He suggested that this data could

be used to cluster bug-fixing activities into redesign efforts for greater debugging efficiency. He suggested that this "megadebugging" approach could increase the dependability growth curve above the curve that would be obtained by one-at-a-time bug fixing.

The fifth problem recounted by Levendel occurred when project managers waited until late in the testing process before making corrections. This practice not only had a cost disadvantage, but it also tended to affect the next product. The imposition of a set of ongoing quality gates (for example, "Is the documentation written?") was used to address this problem.

Next, Levendel cautioned the audience not to assume that software is uniform. Here the testers are well advised to separate the software into smaller, more homogeneous segments.

The seventh problem discussed by Levendel involved excessive reliance on initial fault densities, resulting in a situation in which the actual fault count exceeds the fault counts that would be predicted by extrapolating the initial cumulative fault distribution. This problem was solved by adopting a birth/death reliability model that included a concept of the rate of revisiting an area in the software for repair.

Levendel's eighth problem was the optimistic results that are often obtained from repetitive automated testing when the testing does not account for the operational profile of the in-service system. To solve this problem, AT&T analyzed office data across the USA to create nine canonical operational profiles. Levendel observed that this was an expensive undertaking because of the cost of building a large set of automated tests and the cost of setting everything in the correct initial state for a large environment. In fact, only nine profiles were developed because of cost limitations.

The final problem posed by Levendel concerned the effect of human variability on quality metrics. He exhibited a table showing the number of problems discovered by a group of 51 people. One person discovered 55 problems (48 real faults), while six of the testers found no problems. Levendel noted that he had specifically encouraged the testers in this study to increase their "find" rate; top people had taken the time to visit developers and to learn the code they were testing, thus developing an intuitive feel for design weaknesses. Levendel suggested that this could be used to advantage by understanding software vulnerabilities and focusing testing on these areas.

Levendel concluded his talk with two open issues in achieving software dependability growth through testing. First, he posed a challenge to researchers to create a theory of software instability that identifies design holes and provides techniques and tools to locate them. Last, he lamented the lack of understanding of software architecture and of software testing. The lack of software architecture understanding makes it difficult to predict whether fixes in one module should be accompanied by fixes in other modules. The lack of understanding of software testing is evidenced by the tendency of software in the field to be "trained" by the pattern of use in the field so that defects in infrequently used functions remain as "time bombs" indefinitely.

Levendel ended his talk with a plea that software follow the lead of other industrialized domains by modernizing both evaluation and production techniques.

4.3 Panel Discussion

The panel session opened with short talks by the panelists who had not delivered lectures.

Steve Bunch began by recounting his experiences at Motorola with large amounts of vendor-supplied system software. He indicated that his group could get software with 5,000 - 30,000 bugs and that it takes on the order of two person-days to fix a bug. Thus they had to focus quality improvement efforts on problem functions and critical code. He noted the difficulty of "software archeology," the process of finding the original intent of software by inspecting either the code or the documentation. He also noted that most vendors do not react well to implementing needed changes, especially if the requested changes are not in their product plans. He recounted the experience of the 88 Open organization, which ran the vendors' own test suites and obtained only 60% coverage. Although his group uses various kinds of testing for fault containment, he noted that it is difficult to test in the combinations created by customers. Bunch did observe the "training effect" mentioned by Levendel where software with known defects nevertheless operated with several years of up time.

Michael Dyer of Loral Federal Systems (formerly IBM Federal Systems) prefaced his remarks with a description of the applications implemented by his organization, including space shuttle software, air traffic control, and, more recently, systems for the IRS, FBI and Resolution Trust Corporation. Typically they are large-scale distributed architectures that run 24 hours a day, 7 days a week to serve a diverse set of users. Quality measures include software inspections, a controlled integration process with extensive configuration control, and formal tests to "sell off" requirements to customers. Formal and statistical methods are used selectively. Generally these measures have been successful, resulting in functional acceptance by the customer. Dyer noted that service in the field provides the real data on system reliability. Typical defect rates average 1 defect per 1000 lines of code; this rate has been reduced to as low as 0.01 defects per 1000 lines in special cases in which the customer was willing to spend an unusual amount of money on quality, such as the NASA space shuttle code. Dyer observed a need for better developer knowledge of the real user environment. Also, he stated that the software development process should focus more on total errors instead of discovered errors and needs software error measures other than defects per thousand lines of code. Finally, Dyer noted a need for more formal methods for designing code.

Bliss Jensen of the AT&T Global Communications Division described the software his organization produces for AT&T Definity Systems, a product with about 1.7 million lines of main processor code. New releases of about 100,000 - 500,000 lines of changed code are issued about once a year. Quality requirements are very high, because of the large exposure of the product and the critical role the product plays in businesses such as mail order or reservations. Jensen reported success with an incremental development adaptation of the cleanroom approach. Operational profiles that model the customer environment are used to guide automatic gener-

ators of statistical test cases. Other successful approaches included quality factor assessments and risk-based testing. Less successful efforts included automation of test case execution and the use of process metrics (in contrast to product metrics, which are extensively used.) The principal deficiencies in the system test area include the efficient use of regression tests and the problem of testing interactions with other vendors' products. Software development concerns include last-minute design changes and product breakage, where minor problem fixes result in major outages.

Jacob Abraham of the University of Texas at Austin discussed the three major hardware testing phases: design verification, implementation verification, and manufacturing test. Hardware design verification evaluates whether the design satisfies specific properties. The current practice uses massive amounts of simulation (a superscalar system simulation may extend for billions of clock cycles). Some formal methods are used, typically finite state machine models. Implementation verification also uses formal methods, ranging from approaches that compare high-level and low-level models of finite state machines to theorem proving (in the case of the Viper processor). He observed that most hardware tests are manufacturing tests, which includes structural and functional tests. Structural tests are based on gate-level stuck-at fault models, supported by special built-in-test hardware. He noted that hardware stuck-at models are now being questioned as speeds increase and timing and synchronization faults become more prevalent.

The first question from the audience asked the panelists how they knew when they had reached their quality goals. Jansen replied that features affecting a large number of users are tracked carefully. Products are shipped based on experience with over 40 users and when failure models predict less than one failure per year. Levendel countered that it is impossible to accurately predict the future. Dyer observed that the real confirmation of a system's reliability comes when it is in the field, and that reliability is built up gradually.

In response to a question about experiences using software reengineering tools to analyze as-built software, Bunch noted that it is hard to apply these tools to operating system software such as device drivers. They can be of some use in locating suspect areas of a program. Levendel reported trying many reengineering analysis tools in a limited way, but observed that they have not been very useful. Iyer recalled that the AT&T data analysis project spent about three weeks using tools to analyze failure data, but then spent six months of manual effort to correlate the failures into clusters and associate them with particular functions.

In response to questions about how well universities train software engineers, Iyer replied that much of the material in his lecture is included in a graduate-level course in software testing that supplements two levels of software engineering courses. Levendel lamented that software engineers who construct communications software are trained in software topics such as compiler construction, but have no training in telecommunications. Dyer noted that new hires are smart and write good code, but fail to understand the constraints of software in the bigger picture, where 50-100 other programmers may be involved. Bunch recalled that his employees had learned the importance of testing in the testing class at the University of Illinois.

In response to a question on how system downtime is characterized, Levendel explained that the principal measure is loss of service per office, normalized to 10,000 lines, excluding equipment from other providers and long distance equipment.

In response to a question regarding the effectiveness of embedded tests in maintaining service, Levendel described an experiment in which all software error checking was disabled in switching software that was midway through the development cycle. The system crashed in five minutes or less. Similar treatment of software ready for delivery resulted in degradation in about a week. Jansen noted that the software in the Definity system works in much the same way. He commented that Japanese customers, accustomed to zero defects, were dismayed to see alarm logs with hundreds of entries.

5 The Tools Fair

The afternoon session of Friday, March 25, 1994 was set aside as a tools fair. Attendees were given an opportunity to demonstrate tools or technologies in a non-commercial setting. Because of two unrelated hardware glitches, two of the three demonstrations did not come off as smoothly as they might have. Stephen Cha of the Aerospace Corp. was to demonstrate a tool for safety-critical system analysis. The computer at Aerospace was unavailable to him because of a problem with their Internet link. Stephen gave a short overview of his system instead.

Haim Levendel from AT&T Bell Laboratories was to demonstrate an object-oriented system to develop telephone services. His laboratory computer was also unavailable, necessitating an abbreviated demonstration done without the remote network services.

Lui Sha from the SEI gave a successful, highly praised, demonstration of the "simplex architecture," an architecture designed for safety-critical real-time control systems. About 1/3 of the attendees saw this demonstration.

6 Participants' Comments

All of the participants attending the technology exchange were asked for written comments on the meeting. Over 33% of the attendees provided this feedback. This section summarizes those comments.

Participants completed a form that asked what their expectations about the exchange were and how well these expectations were met. For the most part, the participants' expectations were met (average 4+ out of 5):

- The meeting allowed participants to find out about potentially useful state-of-the-art work in dependable software.
- The discussions helped set an agenda of technology issues that are ripe for further exploration.
- Participants had a chance to meet new colleagues and compare experiences.

In general, participants found the selected topics to be appropriate (average 4+ out of 5), the technical depth of talks was right (average 4+ out of 5), and the format was found to be useful (average 4+ out of 5).

When asked what they liked most and what they liked least about the exchange, the responses were mixed. Some of the attendees really took advantage of the opportunity to meet with others with similar interests. Others mentioned that the format was particularly good for the exchange of information. Each of the sessions was preferred by someone. Individuals selected for special mention included Ram Chillarege (who talked on in-process measurement) and Jacob Abraham (who provided a hardware counterpoint to the Software Testing and Reliability session).

Several attendees liked the fact that the exchange did not concentrate only on DoD-related issues. On the other hand, at least one attendee wished that there had been more direct emphasis on the DoD issues.

Negative comments included a wish that the demonstrations had been held earlier in the exchange. Also, there was a feeling by some that the three topics covered did not play well together. In particular, although in-process measurement and software testing and reliability seem to have some common areas, safety critical systems does not—at least in the eyes of one participant.

Although most attendees who responded thought the presentations were of the right length and depth, there was a minority who thought that they covered the areas at too high a level, and/or were too long—making the speaker less focused.

Nearly all of the respondents would like to see additional exchanges (5- average out of 5).

Most participants would attend additional exchanges and would like to see one held within a year.

Attendees who supplied e-mail addresses have been added to the mailing list "exploder" depend-sw@sei.cmu.edu, which was created to reach the people who attended the first Dependable Software Technology Exchange.

7 Discussion

Although rated a success by the attendees, members of the steering committee and some others feel that the Dependable Software Technology Exchange has not reached its original goal of causing practitioners to learn what is new from the researchers and allowing researchers to learn what doesn't work in practice and what real-world problems must be solved.

The practitioners do attend—although not in the numbers we would like (attendance was off about 15% this year, at least partially due to the cancellation of the Follow-on Early Warning System (FEWS) Project. The problem is that, although the talks were well coordinated this year, the practitioner speakers tend to present success stories rather than telling what research worked and what research was really needed.

Perhaps this can be corrected by better instructions to the practitioner speakers. But the best way for this series of technical exchanges to achieve its goal would be to attract more practitioners with diverse points of view. Although targeted at these practitioners, it seems that some sort of forcing function is needed to get the broad attendance we seek.

For instance, government contract monitors could inform their contractors that two people should be sent to the annual exchange so that the contractors will become more aware of the latest technology. The cost of attending should be billable to the government above the cost of the contract. Similarly, government regulatory agencies (e.g., the FAA, NRC) might encourage people in their industries to attend.

A change of format for the Dependable Software Technology Exchange might also be helpful. The one-hour talks tend to be unfocused. It has been suggested that 45 minutes (without questions) might be a more appropriate length. Also, the panelists, while providing somewhat diverse views, seem to take up too much of the allotted time with their presentations. For the next exchange, it has been proposed that the panelists be given essentially no time for an individual presentation (though they are free to put up a slide or two created in response to the preceding talks). This, combined with lengthening the panel to 90 minutes, should give plenty of time for discussion.

8 List of Attendees

Jacob Abraham
Professor
University of Texas at Austin
ENS 433A
Austin, TX 78712
Phone: (512) 471-8983
Fax: (512) 471-8967
E-Mail: jaa@cerc.utexas.edu

David J. Alberico
Chief, Software Systems Safety
U. S. Air Force
9700 Avenue G, S.E.
Suite 250B
Kirtland AFB, NM 87117-5670
Phone: (505) 846-1172
Fax: (505) 846-2721
E-Mail: alberico%smpts@afsa1.saia.af.mil

Mario R. Barbacci
Program Director
Software Engineering Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213-3890
Phone: (412) 268-7704
Fax: (412) 268-5758
E-Mail: mrb@sei.cmu.edu

Steve Bunch
Chief Scientist
Motorola Inc.
1101 East University Avenue
Urbana, IL 61801
Phone: (217) 384-8515
Fax: (217) 384-8550
E-Mail: srb@urbana.mcd.mot.com

Stephen Cha
Member of Technical Staff
The Aerospace Corporation
2350 E El Segundo Boulevard
El Segundo, CA 90245-4691
Phone: (310) 336-7977
Fax: (310) 336-5833
E-Mail: cha@aero.org

James V. Chelini
Principal Engineer
Raytheon Company
528 Boston Post Road
Sudbury, MA 01776
Phone: (617) 440-3288
Fax: (508) 440-2617
E-Mail: chelinij@tif174.ed.ray.com

Ching-Yun Chen
Senior Engineer
Union Switch & Signal, Inc.
5800 Corporate Drive
Pittsburgh, PA 15237
Phone: (412) 934-2119
Fax: (412) 934-2190
E-Mail: cyc@switch.com

Ram Chillarege
Manager Center for S/W Quality
IBM Corporation
Mail Stop H4 A14
P.O. Box 704
Yorktown Heights, NY 10598
Phone: (914) 784-7596
Fax: (914) 784-6201
E-Mail: ramchill@watson.ibm.com

Elmer Collins
Sr. Member of Technical Staff
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-0830
Phone: (505) 844-1869
Fax: (505) 844-9037

John Cosgrove
President
Cosgrove Computer Systems, Inc.
7411 Earldom Ave.
Phaya Del Rey, CA 90293-8058
Phone: (310) 823-9448
Fax: (310) 821-4021
E-Mail: jdc@textron.com

Michael K. Daskalantonakis
Manager, Software Improvement
Motorola Inc.
Mail Stop: OE321
6501 William Cannon Dr., West
Austin, TX 78735
Phone: (512) 891-2276
Fax: (512) 891-3161
E-Mail: dask@mot.com

Jon Dehn
Sr. Technical Staff Member
Loral Federal Systems
9231 Corporate Boulevard
Rockville, MD 20850
Phone: (301) 640-2912
Fax: (301) 640-3103
E-Mail: dehn@vnet.ibm.com

Mike DeWalt
National Resource Specialist
Federal Aviation Administration
1601 Lind Avenue, S.W.
Renton, WA 98055
Phone: (206) 227-2762
Fax: (206) 227-1181

Jorge L. Diaz-Herrera
Sr. Member of Technical Staff
Software Engineering Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213-3890
Phone: (412) 268-7636
Fax: (412) 268-5758
E-Mail: jldh@sei.cmu.edu

Kevin Driscoll
Staff Scientist
Honeywell Technology Center
3660 Technology Drive
M/S MN65-2500
Minneapolis, MN 55418-1006
Phone: (612) 951-7263
Fax: (612) 951-7438
E-Mail: driscoll@src.honeywell.com

Michael Dyer
Loral Federal Systems
9211 Corporate Boulevard
Rockville, MD 20850
Phone: (301) 640-4211
Fax: (301) 640-4750
E-Mail: mdyer@vnet.ibm.com

Lynn S. Elliott
Manager of Software Engr.
Eli Lilly and Company
4100 North Hamline Avenue
Arden Hills, MN 55112-5798
Phone: (612) 582-2842
E-Mail: elliot_lynn_s@lilly.com

Armen Gabrielian
President
UniView Systems
1192 Elena Privada
Mountain View, CA 94040
Phone: (415) 968-3476
Fax: (415) 968-3476
E-Mail: armen@well.sf.ca.us

Thomas L. Gilchrist
Senior Software Engineer
Boeing Computer Services
P.O. Box 24346
MS 67-ET
Seattle, WA 98124-0346
Phone: (206) 234-4865
Fax: (206) 234-1499
E-Mail: tomg@halcyon.com

Walter Heimerdinger
Fellow
Honeywell Technology Center
3660 Technology Drive
MN 65-2200
Minneapolis, MN 55418-1006
Phone: (612) 951-7332
Fax: (612) 951-7438
E-Mail: walt@src.honeywell.com

M. Frank Houston
Director
Weinberg, Spelton & Sax, Inc.
3029 Brookwood Road
Ellicott City, MD 21042
Phone: (410) 461-6608
Fax: (410) 459-8381
E-Mail: houston@itd.navy.mil

Norman R. Howes
Research Staff Member
Institute for Defense Analyses
1801 N. Beauregard Street
Alexandria, VA 22311
Phone: (703) 845-3533
E-Mail: howes@ida.org

John J. Hudak
Carnegie Mellon University
104 Doray Drive
Pittsburgh, PA 15237
Phone: (412) 268-3368
Fax: (412) 268-6960
E-Mail: jjh@cec.cmu.edu

Michelle M. Hugue
Director, Applied Research
Trident Systems, Inc.
10201 Lee Highway, Ste. 300
Fairfax, VA 22030
Phone: (301) 262-5140
Fax: (703) 273-6608
E-Mail: meesh@nemo.cs.umd.edu

Ravi Iyer
University of Illinois
Coordinated Science Lab.
1308 West Main Street
Urbana, IL 61801
Phone: (217) 333-9732
Fax: (217) 244-5686
E-Mail: iyer@crhc.uiuc.edu

Bliss Jensen
AT&T
11900 Pecos Street
Denver, CO 80234-2703
Phone: (303) 538-4255
Fax: (303) 538-3643
E-Mail: abj@elvis.dr.att.com

Stephen B. Johnson
Research Associate
University of Cincinnati
2912 33rd Avenue, N.E.
St. Anthony, MN 55418
Phone: (612) 781-2898
Fax: (612) 781-3099
E-Mail: john0730@gold.tc.umn.edu

Judith Klein
Senior Programmer
Loral Federal Systems
9231 Corporate Boulevard
Rockville, MD 20850
Phone: (301) 640-2790
Fax: (301) 640-3103

John C. Knight
Professor
University of Virginia
Thornton Hall
University of Virginia
Charlottesville, VA 22903
Phone: (804) 982-2216
Fax: (804) 982-2214
E-Mail: knight@virginia.edu

Jari Koistinen
Ellendel/Stockholm University
Ellendel Box 1505
12525 Atusjo
Stockholm,
SWEDEN
Phone: +46-8-7274359
Fax: +46-8-7274220
E-Mail: euajak@cua.vicsson.se

Gary Koob
Chief of Naval Research
U. S. Navy
800 N. Quincy Street
Ballston Tower One
Arlington, VA 22217-5660
Phone: (703) 696-0872
Fax: (703) 696-0934
E-Mail: koob@itd.nrl.navy.mil

Y. Haim Levendel
Technology Development Director
AT&T Bell Laboratories
263 Shuman Boulevard
Room 1Z-108
Naperville, IL 60566-7050
Phone: (708) 979-1310
Fax: (708) 979-1434
E-Mail: yhl@ihlpl.att.com

David R. Luginbuhl
Software & Sys. Program Manager
Air Force Office of Scientific Research
AFOSR/NM
110 Duncan Avenue, Ste. B115
Bolling AFB, DC 20332
Phone: (202) 767-5028
Fax: (202) 404-7496
E-Mail: luginbuh@afosr.af.mil

Michael R. Lyu
Member of Technical Staff
Bellcore
445 South Street
Morristown, NJ 07960-1910
Phone: (201) 829-3999
Fax: (201) 829-5981
E-Mail: lyu@bellcore.com

Joseph R. Marshall
Advisory Engineer
Loral Federal Systems
9500 Godwin Drive, MS 018
Manassas, VA 22110
Phone: (703) 367-9411
Fax: (703) 367-9440
E-Mail: joe_marshall@vnet.ibm.com

Roy Maxion
System Scientist
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
Phone: (412) 268-7556
Fax: (412) 681-5739
E-Mail: maxion@cs.cmu.edu

John A. McDermid
University of York
Heslington
York, North Yorkshire, YO1 5DD
UNITED KINGDOM
Phone: +44-0-904-432782
Fax: +44-0-904-432708
E-Mail: jam@minster.york.ac.uk

Cuong Nguyen
Consulting Engineer
Union Switch & Signal, Inc.
5800 Corporate Drive
Pittsburgh, PA 15237
Phone: (412) 934-2184
E-Mail: ccn@switch.com

Suman Purwar
Assistant Professor
Wheeling Jesuit College
316 Washington Avenue
Wheeling, WV 26003
Phone: (304) 243-2340
E-Mail: sumanp@acc.wjc.edu

Arthur Robinson
President
System Technology Development Corp
1035 Sterling Road, Ste. 101
Herndon, VA 22070
Phone: (703) 478-0687
Fax: (703) 478-0689
E-Mail: wk01811@worldlink.com

John Rushby
Program Director
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025
Phone: (415) 859-5456
Fax: (415) 859-2844
E-Mail: rushby@csl.sri.com

Richard C. Scalzo
Mathematician
Naval Surface Warfare Center
Code A44
10901 New Hampshire Ave.
Silver Spring, MD 20903-5640
Phone: (301) 394-2926
Fax: (301) 394-1164
E-Mail: rscalzo@relay.nswc.navy.mil

Fred Schneider
Professor
Cornell University
Upson Hall
Ithaca, NY 14853
Phone: (607) 255-9221
E-Mail: fbs@cs.cornell.edu

Lui Sha
Project Leader
Software Engineering Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213-3890
Phone: (412) 268-5875
Fax: (412) 268-5758
E-Mail: lrs@sei.cmu.edu

Bassam Shanti
Assistant Professor
Wheeling Jesuit College
316 Washington Avenue
Wheeling, WV 26003
Phone: (304) 243-2302

Roger Sherman
Director of Testing
Microsoft Corporation
1 Microsoft Way
Redmond, WA 98052
Phone: (206) 936-3447
E-Mail: rogersh@microsoft.com

Daniel P. Siewiorek
Professor
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213-3890
Phone: (412) 268-2570
Fax: (412) 681-5739
E-Mail: dps@cs.cmu.edu

Mark I. Snyder
Space Systems Software Engineer
Phillips Lab
VTES (Space Software Concepts)
Kirtland AFB, NM 87117
Phone: (505) 246-8986
Fax: (505) 246-2290
E-Mail: snyderm@plk.af.mil

Steven Spielman
Principal Engineer
Raytheon Company
528 Boston Post Road
Sudbury, MA 01776
Phone: (508) 490-2352
Fax: (508) 440-2337
E-Mail: steven_i_spielman@ccmail.ed.ray.com

Jay K. Strosnider
Associate Professor
Carnegie Mellon University
5000 Forbes Avenue
Porter Hall B17
Pittsburgh, PA 15213-3890
Phone: (412) 268-6927
Fax: (412) 268-3890
E-Mail: jks@usa.ece.cmu.edu

Neeraj Suri
MTS
Allied Signal Research Center
9140 Old Annapolis Road
Columbia, MD 21045
Phone: (410) 964-4157
Fax: (410) 992-5813
E-Mail: suri@batc.allied.com

Robert S. Swarz
Director, Reliability & Main. Ctr.
The MITRE Corporation
202 Burlington Road
Bedford, MA 01730-1420
Phone: (617) 271-2847
Fax: (508) 358-1124
E-Mail: rswarz@mitre.org

Gary J. Vecellio
Lead Scientist
The MITRE Corporation
7525 Colshire Drive, MS W624
McLean, VA 22102
Phone: (703) 883-5696
Fax: (703) 883-1339
E-Mail: vecellio@mitre.org

Doiores R. Wallace
Computer Scientist
National Institute of Stds. & Tech.
Technology Building, Rm. B266
Gaithersburg, MD 20899
Phone: (301) 975-3340
Fax: (301) 926-3696
E-Mail: wallace@sst.ncsl.nist.gov

Chris Walter
Manager, Systems
Allied-Signal
9140 Route 108
Columbia, MD 21045
Phone: (410) 964-4082
Fax: (410) 992-5813
E-Mail: chris@batc.allied.com

Charles B. Weinstock
Sr. Member of the Technical Staff
Software Engineering Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213-3890
Phone: (412) 268-7719
Fax: (412) 268-5758
E-Mail: weinstock@sei.cmu.edu

Tom Wheeler
Member of Technical Staff
The MITRE Corporation
202 Burlington Rd., M/S B244
Bedford, MA 01730-1420
Phone: (617) 271-2589
Fax: (617) 271-2911
E-Mail: twheeler@mitre.org

Bing Yu
Senior Engineer
Union Switch & Signal, Inc.
5800 Corporate Drive
Pittsburgh, PA 15237
Phone: (412) 934-2164
Fax: (412) 934-2190
E-Mail: bingyu@switch.com

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified			1b. RESTRICTIVE MARKINGS None		
2a. SECURITY CLASSIFICATION AUTHORITY N/A			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release Distribution Unlimited		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A					
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CMU/SEI-94-SR-6			5. MONITORING ORGANIZATION REPORT NUMBER(S)		
6a. NAME OF PERFORMING ORGANIZATION Software Engineering Institute		6b. OFFICE SYMBOL (if applicable) SEI	7a. NAME OF MONITORING ORGANIZATION SEI Joint Program Office		
6c. ADDRESS (city, state, and zip code) Carnegie Mellon University Pittsburgh PA 15213			7b. ADDRESS (city, state, and zip code) HQ ESC/ENS 5 Eglin Street Hanscom AFB, MA 01731-2116		
8a. NAME OFFUNDING/SPONSORING ORGANIZATION SEI Joint Program Office		8b. OFFICE SYMBOL (if applicable) ESC/ENS	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F1962890C0003		
8c. ADDRESS (city, state, and zip code)) Carnegie Mellon University Pittsburgh PA 15213			10. SOURCE OF FUNDING NOS.		
			PROGRAM ELEMENT NO 63756E	PROJECT NO. N/A	TASK NO N/A
11. TITLE (Include Security Classification) Second Dependable Software Technology Exchange			15. PAGE COUNT 40 pp.		
12. PERSONAL AUTHOR(S) Charles B. Weinstock, SEI and Walter Heimerdinger, Honeywell Technology Center					
13a. TYPE OF REPORT Final		13b. TIME COVERED FROM TO		14. DATE OF REPORT (year, month, day) June 1994	
16. SUPPLEMENTARY NOTATION					
17. COSATI CODES			18. SUBJECT TERMS (continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB. GR.	dependable real-time software Safety Critical		
			Dependable Software Technology Exchange Testing		
			Improcess Measurement		
19. ABSTRACT (continue on reverse if necessary and identify by block number) On March 24 and 25, 1994, the Open Attribute Engineering Project hosted the Second Dependable Software Technology Exchange. The exchange, sponsored by the Air Force Phillips Laboratories, the Office of Naval Research, and the Air Force Space and Missile Systems Center, brought together researchers and system developers, providing an opportunity for the researchers to learn the needs of the developers and for the developers to learn about techniques being investigated by the researchers. This report summarizes what transpired at the meeting. <div style="text-align: right;">(please turn over)</div>					
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED <input checked="" type="checkbox"/> SAME AS RPT <input type="checkbox"/> DTIC USERS <input type="checkbox"/>			21. ABSTRACT SECURITY CLASSIFICATION Unclassified, Unlimited Distribution		
22a. NAME OF RESPONSIBLE INDIVIDUAL Thomas R. Miller, Lt Col, USAF			22b. TELEPHONE NUMBER (include area code) (412) 268-7631		22c. OFFICE SYMBOL ESC/ENS (SEI)

ABSTRACT — continued from page one, block 19